**INTRODUCTION:**

The Travel Agency Management System is designed to streamline and automate the core operations of a travel agency. This system manages the entire process of booking travel packages, customer interactions, and payment handling. It enables travel agents to efficiently organize customer details, available travel packages, bookings, itineraries, and payment transactions, all in one integrated platform.

The system consists of several key modules:

* **Customer Management**: Stores detailed customer information, including contact details and booking history, ensuring personalized services and improved customer relationships.
* **Travel Package Management**: Organizes and manages travel packages offered by the agency, including package details, prices, and durations.
* **Booking Management**: Tracks customer bookings, including the package chosen, booking date, number of travelers, and booking status (Confirmed, Pending, or Cancelled).
* **Itinerary Management**: Provides detailed day-by-day itineraries, helping customers understand their travel schedule and planned activities.
* **Payment Management**: Ensures secure handling of payments, recording payment methods, amounts, and dates for each booking.

This system enhances the efficiency of travel agencies by centralizing all relevant data, improving operational workflows, and enabling smooth interactions between customers, travel agents, and service providers. The implementation of this system ensures that all aspects of travel management are handled accurately and promptly, providing an enhanced customer experience and supporting the growth of the agency.

**AIM of the project**

The travel agency management system facilitates customers to inquire about travel packages, book trips, manage itineraries, and make payments. The aim of this project is to design and develop a database to maintain records of travel packages, customer bookings, itineraries, and payments. This system ensures proper tracking and management of customer transactions and travel plans.

**DESCRIPTION**

The system allows customers to book travel packages and manage their itineraries. A customer can inquire about available travel packages and choose one based on their preferences. Before booking, the availability of the package and required accommodations are verified. Upon successful booking, the details are stored, and a unique booking ID is generated. Customers can also cancel their bookings, which updates the system and makes the slots available for other customers.

Key functionalities include:

1.Managing records of travel packages, customers, and payments.

2.Handling booking and cancellation processes.

3.Generating itineraries for booked travel packages.

4.Maintaining payment details for transactions.

**LIST OF ASSUMPTIONS**

To simplify the implementation of the travel agency management system, the following assumptions have been made:

The number of travel packages is restricted to 10.

The booking is only available for travel packages scheduled within the next 30 days.

A customer can book only one package per transaction.

Payment modes include credit card, PayPal, and bank transfer.

Cancellation is allowed up to 48 hours before the package start date.

**DESCRIPTION OF TABLES AND PROCEDURES**

1. **TravelPackages**

Purpose: Stores details of all available travel packages.

Columns:

PackageID: Unique identifier for each package(primary key).

Name: Name of the travel package.

Description: Brief description of the package.

Price: Cost of the package.

Duration: Number of days.

AvailableDates: Dates on which the package is available.

Constraint: The PackageID is unique.

2. **Customers**

Purpose: Stores customer details.

Columns:

CustomerID: Unique identifier for each customer(primary key).

Name: Name of the customer.

Email: Email address of the customer.

Phone: Contact number.

Constraint: The CustomerID is unique.

3. **Bookings**

Purpose: Manages booking records.

Columns:

BookingID: Unique identifier for each booking.

CustomerID: Refers to the customer who made the booking.(Foreign key)

PackageID: Refers to the travel package booked.

BookingDate: Date of booking.

Status: Status of the booking (Confirmed or Cancelled).

Constraints:

CustomerID must exist in the Customers table.

PackageID must exist in the TravelPackages table.

4. **Itineraries**

Purpose: Stores details of the itinerary for each booking.

Columns:

ItineraryID: Unique identifier for each itinerary entry.

BookingID: Refers to the booking associated with the itinerary.

Day: Day of the trip.

Activity: Activity planned for the day.

Location: Location of the activity.

Constraint: BookingID must exist in the Bookings table.

5. **Payments**

Purpose: Manages payment details for bookings.

Columns:

PaymentID: Unique identifier for each payment.

BookingID: Refers to the booking associated with the payment.

PaymentDate: Date of payment.

Amount: Amount paid.

Method: Payment method (Credit Card, PayPal, etc.).

Constraint: BookingID must exist in the Bookings table.

PROCEDURES

1**. BookTrip**

Input: CustomerID, PackageID, DesiredDate.

Process:

Validate PackageID and DesiredDate.

Check availability of the package on the selected date.

If available, insert details into the Bookings table with Confirmed status.

Generate a BookingID and store it.

Insert the itinerary into the Itineraries table.

Output: Booking confirmation with BookingID.

2. **CancelTrip**

Input: BookingID.

Process:

Validate BookingID.

If valid, delete the record from the Bookings table.

Update the availability status of the package in the TravelPackages table.

Remove associated entries from the Itineraries table.

Output: Cancellation confirmation.

3. **ProcessPayment**

Input: BookingID, PaymentMethod, Amount.

Process:

Validate BookingID.

Insert payment details into the Payments table.

Update the booking status to Paid.

Output: Payment receipt with PaymentID.

**Relationships**

1. Customers → Bookings:

Relationship Name: **Makes.**

**Cardinality:** A customer can make multiple bookings (1:N). A booking is always associated with exactly one customer.

Cardinality Ratio: 1:N.

**Participation**:

Customers: Total Participation (Every booking must be made by a customer).

Bookings: Partial Participation (Not all customers necessarily make bookings).

2. TravelPackages → Bookings:

Relationship Name: **ReservedFor.**

**Cardinality**: A travel package can be reserved in multiple bookings (1:N). A booking must be associated with exactly one package.

Cardinality Ratio: 1:N.

**Participation:**

TravelPackages: Partial Participation (Not all packages are necessarily booked).

Bookings: Total Participation (Every booking must correspond to a travel package).

3. TravelPackages → Itineraries:

Relationship Name: **Has**.

Cardinality: A travel package can have multiple itinerary entries (1:N). An itinerary must belong to exactly one travel package.

Cardinality Ratio: 1:N.

**Participation:**

TravelPackages: Total Participation (Every itinerary belongs to a travel package).

Itineraries: Total Participation (An itinerary entry cannot exist without a travel package).

4. Bookings → Payments:

Relationship Name: **HasPayment**.

Cardinality: A booking can have multiple payments (1:N). A payment must correspond to exactly one booking.

Cardinality Ratio: 1:N.

**Participation:**

Bookings: Partial Participation (Not all bookings may have payments).

Payments: Total Participation (Every payment corresponds to a booking).

**Differentiating Strong and Weak Entities**:

Strong Entity Types:

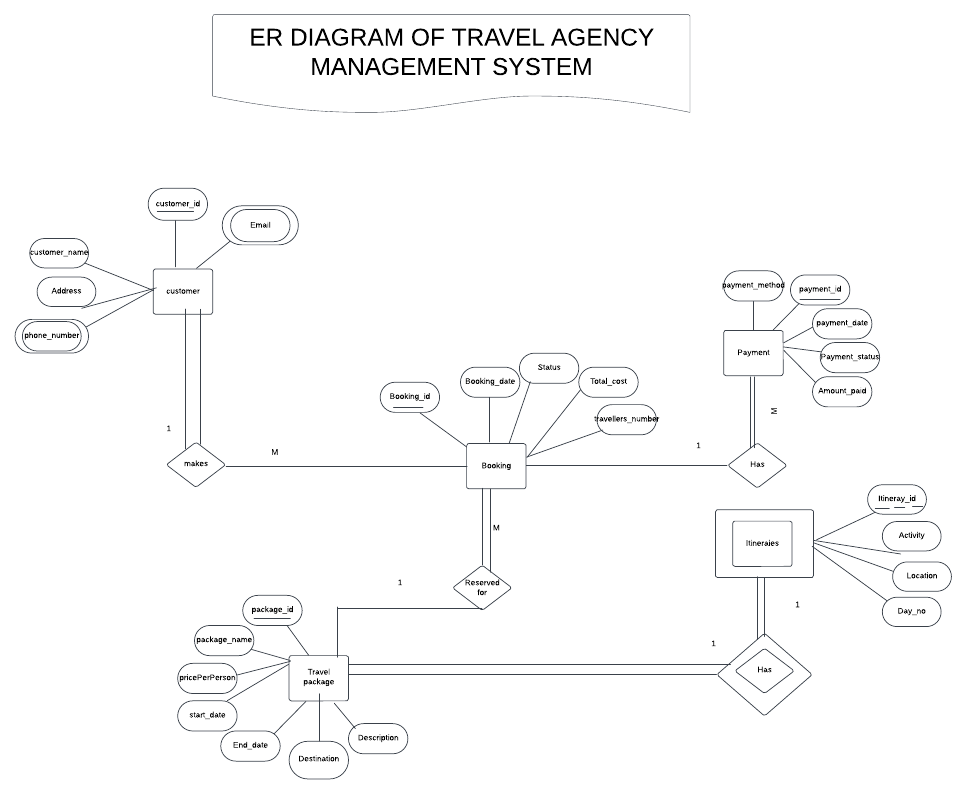
Customers, TravelPackages, Bookings, and Payments are strong entities as they have their own unique primary keys.

Weak Entity Type:

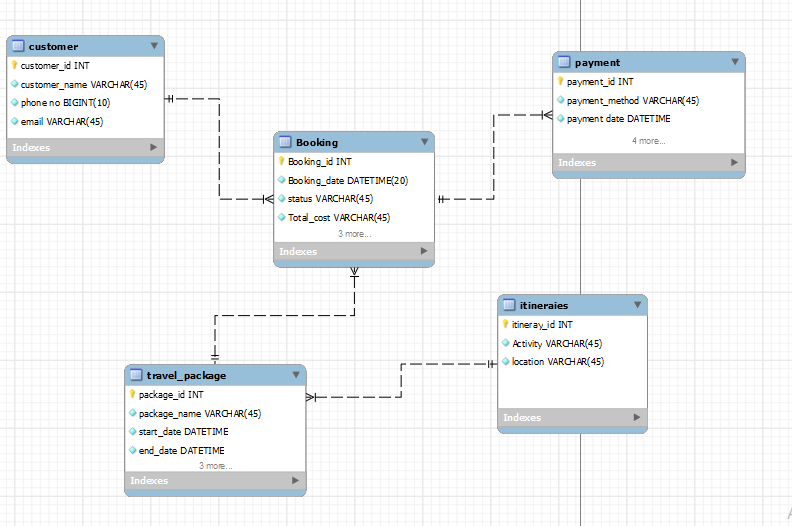
Itineraries:

Itinerary entries depend on a specific travel package (identified by PackageID).

Weak Relationship: The "Has" relationship between TravelPackages and Itineraries is a weak relationship, as Itineraries depends on TravelPackages for identification.



**ER using workbench**

****

**Create Relation by using the above ER Diagram**

CREATE TABLE Customers (

CustomerID varchar(4) PRIMARY KEY,

c\_Name VARCHAR(100) NOT NULL,

Email VARCHAR(150) NOT NULL UNIQUE,

Phone CHAR(15) NOT NULL,

Address CHAR(25)

);

CREATE TABLE TravelPackages (

PackageID varchar(4) PRIMARY KEY,

p\_Name VARCHAR(100) NOT NULL,

PackageDescription TEXT,

Priceperperson DECIMAL(8, 2) NOT NULL,

Duration TINYINT NOT NULL, -- in days

Start\_Date date,

End\_Date date

);

CREATE TABLE Bookings (

BookingID varchar(4) PRIMARY KEY,

CustomerID varchar(4) NOT NULL,

PackageID varchar(4) NOT NULL,

BookingDate DATETIME NOT NULL,

BookingStatus VARCHAR(20) CHECK (BookingStatus IN ('Confirmed', 'Pending', 'Cancelled')),

TotalCost DECIMAL(10, 2),

TravelersNumber INT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (PackageID) REFERENCES TravelPackages(PackageID)

);

CREATE TABLE Itineraries (

ItineraryID varchar(4) PRIMARY KEY,

BookingID varchar(4) NOT NULL,

Days TINYINT NOT NULL, -- Day of the itinerary

Activity VARCHAR(255) NOT NULL,

Location VARCHAR(255),

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

CREATE TABLE Payments (

PaymentID varchar(4) PRIMARY KEY,

BookingID varchar(4) NOT NULL,

PaymentDate DATE NOT NULL,

Amount DECIMAL(10,2) NOT NULL DEFAULT 0.00,

Method VARCHAR(50) CHECK (Method IN ('Credit Card', 'PayPal', 'Bank Transfer', 'No Payment')),

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

**SAMPLE(not all data) INSERT DATA INTO RELATIONS:**

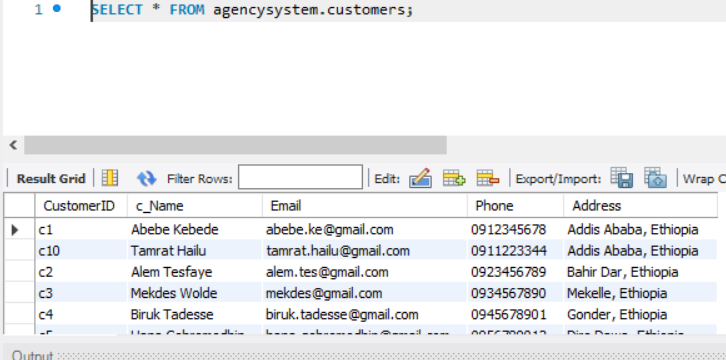
We inserted 10 rows for each relation but for sample use few of them.

**INSERT INTO** Customers (CustomerID, c\_Name, Email, Phone, Address) VALUES

('c1', 'Abebe Kebede', 'abebe.ke@gmail.com', '0912345678', 'Addis Ababa, Ethiopia')

('c2', 'Alem Tesfaye', 'alem.tes@gmail.com', '0923456789', 'Bahir Dar, Ethiopia');

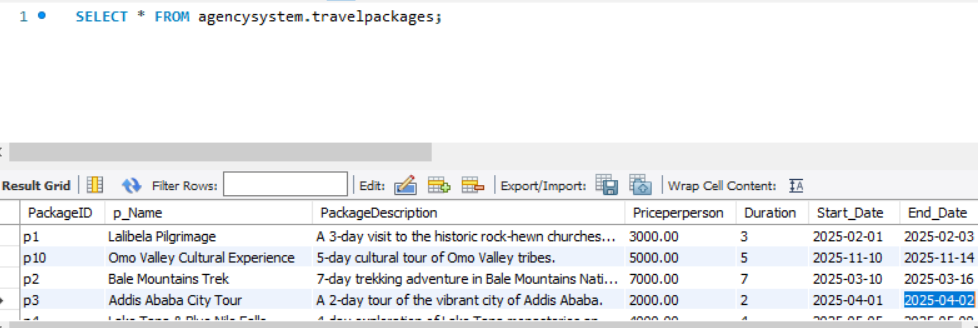
………..



**INSERT INTO TravelPackages (PackageID, p\_Name, PackageDescription, Priceperperson, Duration, Start\_Date, End\_Date) VALUES**

**('p1', 'Lalibela Pilgrimage', 'A 3-day visit to the historic rock-hewn churches of Lalibela.', 3000.00, 3, '2025-02-01', '2025-02-03'),**

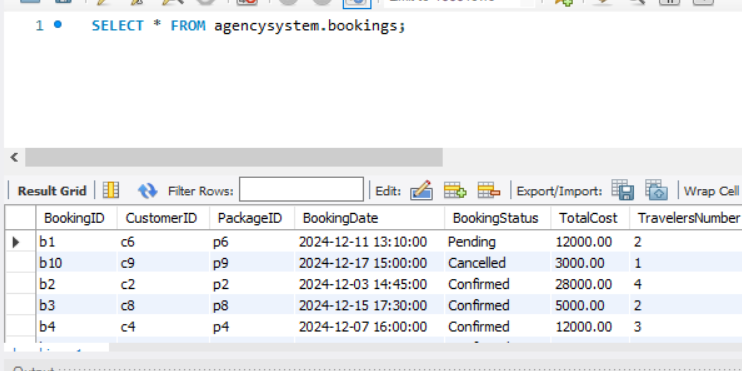
**('p2', 'Bale Mountains Trek', '7-day trekking adventure in Bale Mountains National Park.', 7000.00, 7, '2025-03-10', '2025-03-16');**

****

**INSERT INTO Bookings (BookingID, CustomerID, PackageID, BookingDate, BookingStatus, TotalCost, TravelersNumber) VALUES**

**('b5', 'c1', 'p1', '2024-12-01 10:30:00', 'Confirmed', 6000, 2),**

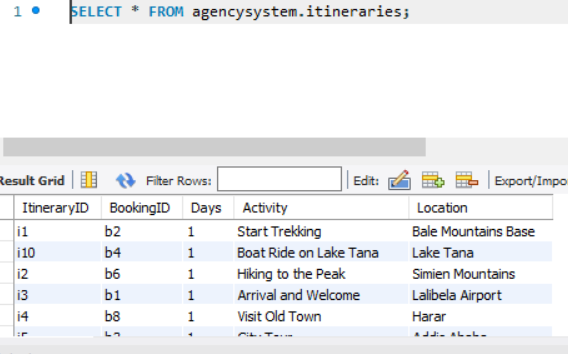
**('b2', 'c2', 'p2', '2024-12-03 14:45:00', 'Confirmed', 28000, 4);**



**INSERT INTO Itineraries (ItineraryID, BookingID, Days, Activity, Location) VALUES**

**('i3', 'b1', 1, 'Arrival and Welcome', 'Lalibela Airport'),**

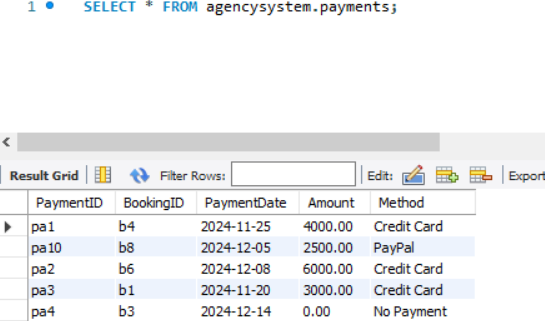
**('i7', 'b1', 2, 'Visit Rock-hewn Churches', 'Lalibela');**

****

**INSERT INTO Payments (PaymentID, BookingID, PaymentDate, Amount, Method) VALUES**

**('pa3', 'b1', '2024-11-20', 3000.00, 'Credit Card'),**

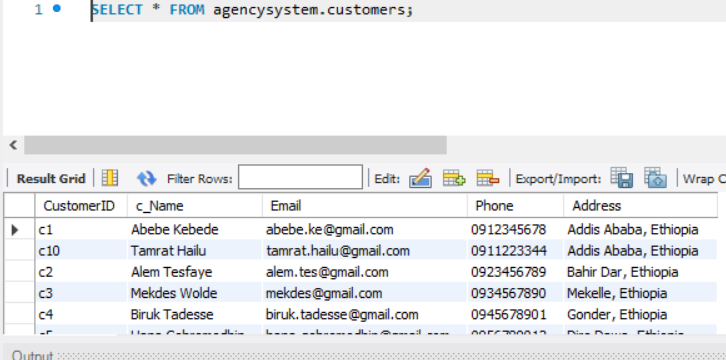
**('pa7', 'b2', '2024-11-22', 7000.00, 'PayPal');**



### Query 1: View All Customers

SQL Statement**:**

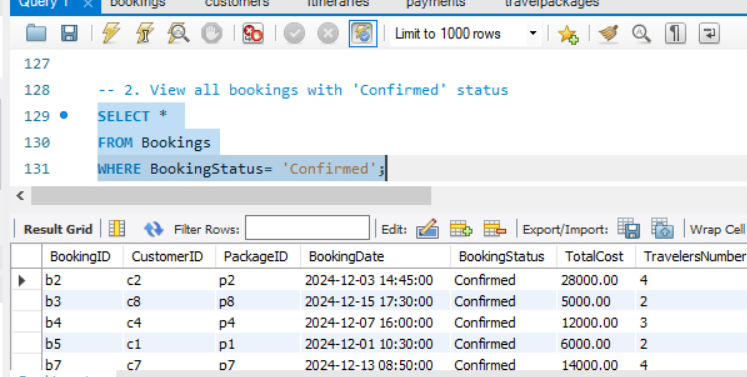
**SELECT \* FROM Customers;**

****

**Explanation:** This query retrieves all records from the Customers table, displaying customer details such as name, email, and phone number.

### Query 2: View Confirmed Bookings

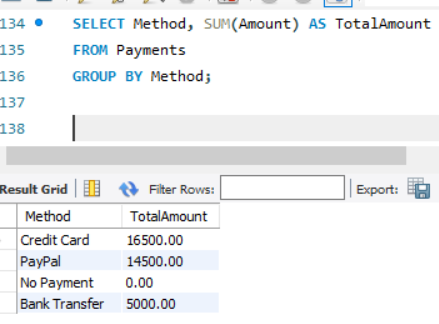
**SQL Statement: SELECT \* FROM Bookings WHERE BookingStatus = 'Confirmed';**



**Explanation:** Filters the Bookings table to display only records where the booking status is "Confirmed."

### Query 3: Total Payment Amounts by Method

**SQL Statement: SELECT Method, SUM(Amount) AS TotalAmount FROM Payments GROUP BY Method;**



**Explanation:** Groups payments by method (e.g., Credit Card, PayPal) and calculates the total amount paid for each method.

### Query 4: Customer and Package Details for Each Booking

**SQL Statement:**

**SELECT**

**b.BookingID,**

**c.c\_Name AS CustomerName,**

**p.p\_Name AS PackageName,**

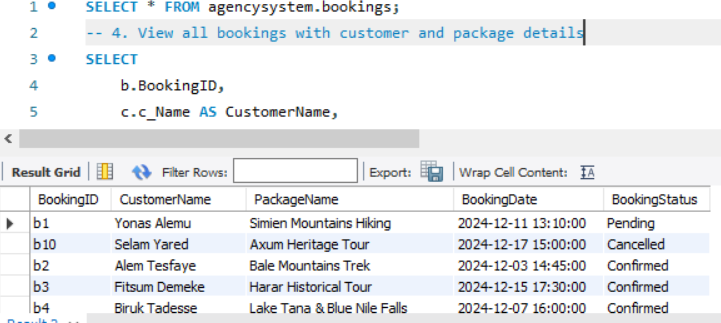
**b.BookingDate,**

**b.BookingStatus**

**FROM Bookings b**

**JOIN Customers c ON b.CustomerID = c.CustomerID**

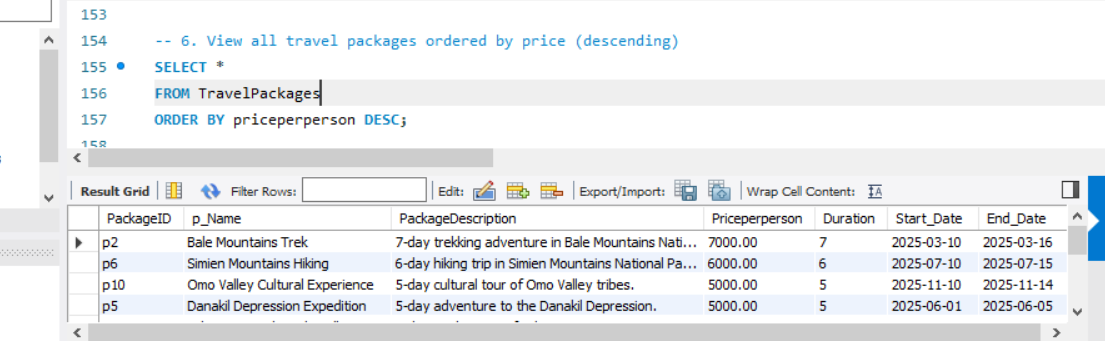
**JOIN TravelPackages p ON b.PackageID = p.PackageID;**



**Explanation:** Joins the Bookings, Customers, and TravelPackages tables to display detailed information about bookings, including customer name, package name, booking date, and status.

### Query 5: View All Travel Packages Ordered by Price

**SQL Statement: SELECT \* FROM TravelPackages ORDER BY Priceperperson DESC;**



**Explanation:** Retrieves all travel packages and sorts them in descending order based on their price, displaying the most expensive packages first.

### Query 7: View Customers with Pending Bookings and Associated Package Details

**SQL Statement:**

SELECT

c.c\_Name,

b.BookingDate,

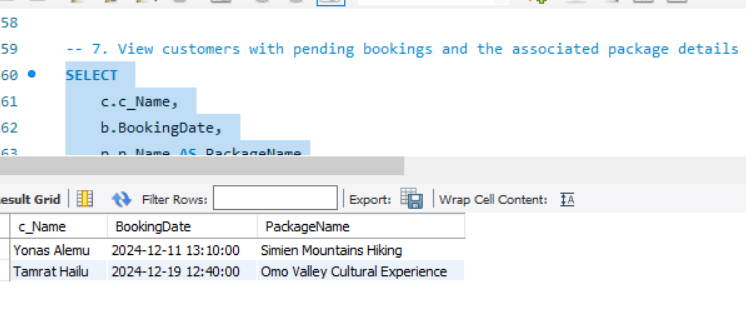
p.p\_Name AS PackageName

FROM Bookings b

JOIN Customers c ON b.CustomerID = c.CustomerID

JOIN TravelPackages p ON b.PackageID = p.PackageID

WHERE b.BookingStatus = 'Pending';



**Explanation:** Filters the Bookings table to identify customers with "Pending" bookings, along with the booking date and package name.

### Query 8: View Total Earnings for Each Travel Package

**SQL Statement:**

SELECT

p.p\_Name AS PackageName,

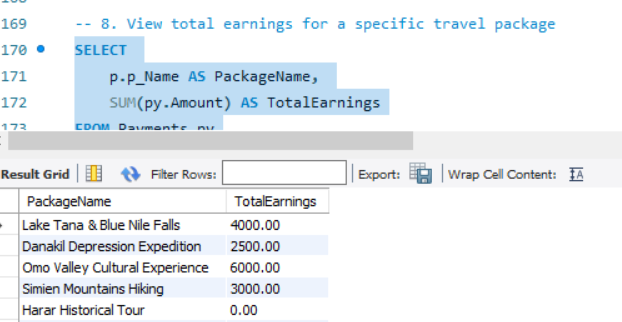
SUM(py.Amount) AS TotalEarnings

Payments py

JOIN FROM Bookings b ON py.BookingID = b.BookingID

JOIN TravelPackages p ON b.PackageID = p.PackageID

GROUP BY p.PackageID;



**Explanation:** Groups payments by travel package and calculates the total earnings for each package by summing the payment amounts.

**Additional queries a travel agency want:**

1. View all travel packages ordered by price (descending)

SELECT \*

FROM TravelPackages

ORDER BY priceperperson DESC;

2. View the most booked travel packages

SELECT

p.p\_Name AS PackageName,

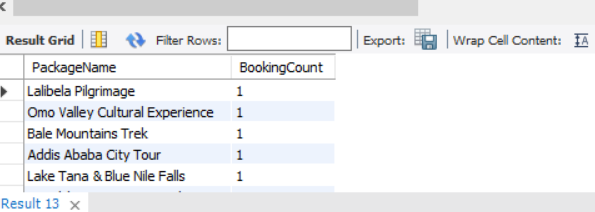
COUNT(b.PackageID) AS BookingCount

FROM Bookings b

JOIN TravelPackages p ON b.PackageID = p.PackageID

GROUP BY b.PackageID

ORDER BY BookingCount DESC;

****

**3. View customers who have made no payments**

**SELECT**

**c.c\_Name AS CustomerName,**

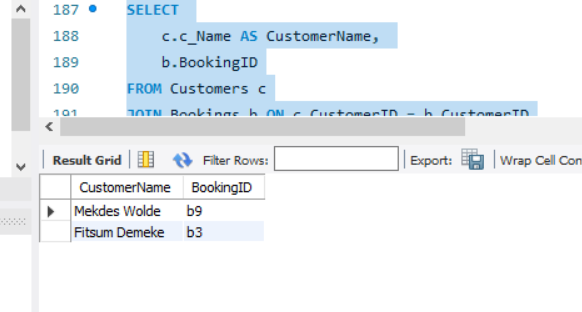
**b.BookingID**

**FROM Customers c**

**JOIN Bookings b ON c.CustomerID = b.CustomerID**

**LEFT JOIN Payments py ON b.BookingID = py.BookingID**

**WHERE py.Amount IS NULL OR py.Amount = 0;**

****

4. View total number of bookings per status

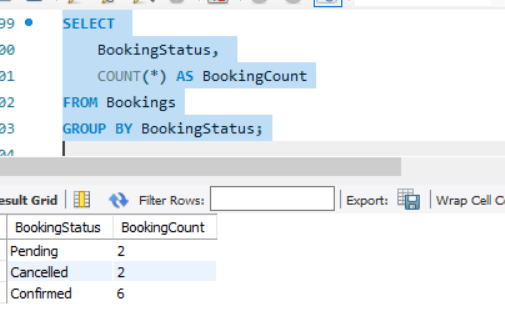
SELECT

BookingStatus,

COUNT(\*) AS BookingCount

FROM Bookings

GROUP BY BookingStatus**;**

****

. View the most active customers based on the number of bookings

SELECT

c.c\_Name AS CustomerName,

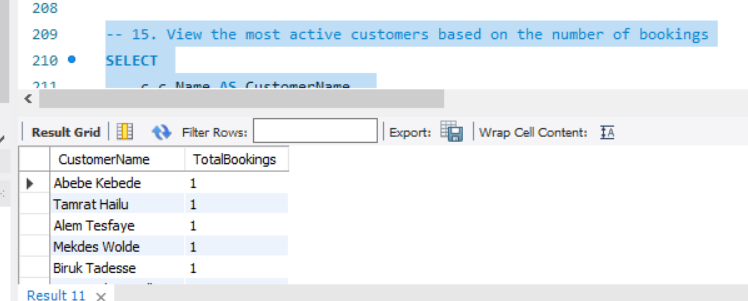
COUNT(b.BookingID) AS TotalBookings

FROM Customers c

JOIN Bookings b ON c.CustomerID = b.CustomerID

GROUP BY c.CustomerID

ORDER BY TotalBookings DESC;



## 3. Challenges we  Encountered and Solutions

### 1: Ensuring Data Integrity

* **Issue:** Maintaining consistency across tables with multiple foreign key relationships.
* **Solution:** Implemented FOREIGN KEY constraints to enforce referential integrity. For example, CustomerID in Bookings references Customers.

### 2: Handling Payment Defaults

* **Issue:** Distinguishing between unpaid bookings and bookings with no payment required.
* **Solution:** Added a default Amount of 0.00 for "No Payment" and included this logic in queries.

### 3: Query Optimization

* **Issue:** Slow performance when retrieving large datasets.
* **Solution:** Added indexes on frequently used columns like CustomerID, PackageID, and BookingID to improve query performance.

### 4: Validating Data Formats

* **Issue:** Inconsistent email and phone number formats.
* **Solution:** Established stricter validation rules during data entry and updates.

**5.** To implement the new scenario where the Amount for the payment is automatically set to match the **TotalCost** from the **Bookings table.**scenario have not been solved because out of scope so we insert calculated total paid= no of traveller\*priceperperson but most project in Github use this: CREATE TRIGGER set\_payment\_amount BEFORE INSERT ON Payments into attribute amount FOR EACH ROW

## Conclusion

The Travel Agency Database Management System successfully streamlines the management of customers, travel packages, bookings, itineraries, and payments. By addressing key challenges and optimizing the schema, the system ensures efficient operation and scalability. Future enhancements could include implementing advanced analytics and integrating user-friendly front-end interfaces.